Master's Thesis

Fine-grained Memory Locality Aware Scheduling Techniques in a Virtualized Cluster

Hyungoo Kim

Department of Computer Science Engineering

Graduate School of UNIST

2018

Fine-grained Memory Locality Aware Scheduling Techniques in a Virtualized Cluster

Hyungoo Kim

Department of Computer Science Engineering

Graduate School of UNIST

Fine-grained Memory Locality Aware Scheduling Techniques in a Virtualized Cluster

A thesis/dissertation
submitted to the Graduate School of UNIST
in partial fulfillment of the
requirements for the degree of
Master of Science

Hyungoo Kim

12/12/2017
Approved by

_____

Advisor
Young-ri. Choi

Fine-grained Memory Locality Aware Scheduling Techniques in a Virtualized Cluster

Hyungoo Kim

This certifies that the thesis of Hyungoo Kim is approved.

12/12/2017

signature

_____

Advisor: Young-ri. Choi

signature

_____

Beomseok Nam: Thesis Committee Member #1

signature

_____

Woongki Baek: Thesis Committee Member #2

Abstract

Big-data processing platforms can be used in a virtualized cluster. But, the well-known big data processing system does not take account into the virtualized cluster. In this research, we designed and implemented *Cache-Affinity and Virtualization-Aware (CAVA) resource manager* which schedules tasks considering both in-memory cache and virtualized environment.

CAVA uses the properties of a virtualized cluster such that transferring data between two co-resident Virtual Machines is performed very fast because it does not use physical network IO. CAVA also tries to read input data from in-memory cache prior to disk regardless of whether input data is in the same or different physical machine. This methodology is very effective in the virtualized cluster because it exploits the chances to reuse cached input data in the cluster and mitigate disk I/O of physical machines. We implemented CAVA on the top of the Hadoop and we showed that CAVA improves overall performance by 34.2% on average over multiple workloads compared to the original Hadoop.

Contents

List of Figures

List of Tables

1. Introduction

Cloud computing environment makes it easy to manage resources by supporting automated provisioning. So, running Hadoop in virtualized environment became very common. But, existing Hadoop distributions do not consider the virtualized environment where multiple VMs share computing resources on a single physical machine.

Current Hadoop distributions support only three kinds of data localities. Those are *data-local*, *rack-local*, and *off-rack*. Although the newest Hadoop supports the distributed in-memory cache feature, it does not consider in-memory cache when a Hadoop ResourceManager schedules resources using data localities.

But, there are extra data localities we can find in the virtualized cluster and we can optimize the performance of MapReduce applications by making Hadoop consider those data localities and the in-memory cache in the virtualized cluster. If two VMs are in the same physical machine, they do not use network bandwidth when they transfer data from one to another. So, reading an input data from the in-memory cache of the VM which is in the same physical machine can avoid both network communication and disk I/O. Although the cached input data is in the different physical machine, a task of the MapReduce application still can avoid disk I/O by reading the input data from the in-memory.

Assume that VM v1 has an input block A of an incoming task in the local in-memory, but v1 does not afford to run the task and co-resident VM v2 affords to run the task. In this case, we assign the task to v2 so that task can reuse cached input data from v1.

In this study, we investigate additional data localities from which Hadoop can take performance gain. Especially, we exploit cached input data in the same physical machine or in the same rack.

The main contributions of this paper are as follows.

1. We subdivide the data locality because legacy Hadoop does not consider both VM topology in the virtualized cluster and in-memory cache.
2. We introduce physical machine memory locality and rack machine memory locality which gives a better scheduling chance to improve performance by making a task read input data from in-memory cache even though cached data is not in a local VM.
3. We propose scheduling techniques which fully utilize additional data localities based on the VM topology and we demonstrate that our scheduling technique improves the performance of MapReduce applications significantly because it improves hit ratio of in-memory cache and reduces disk I/O.

2. System Design

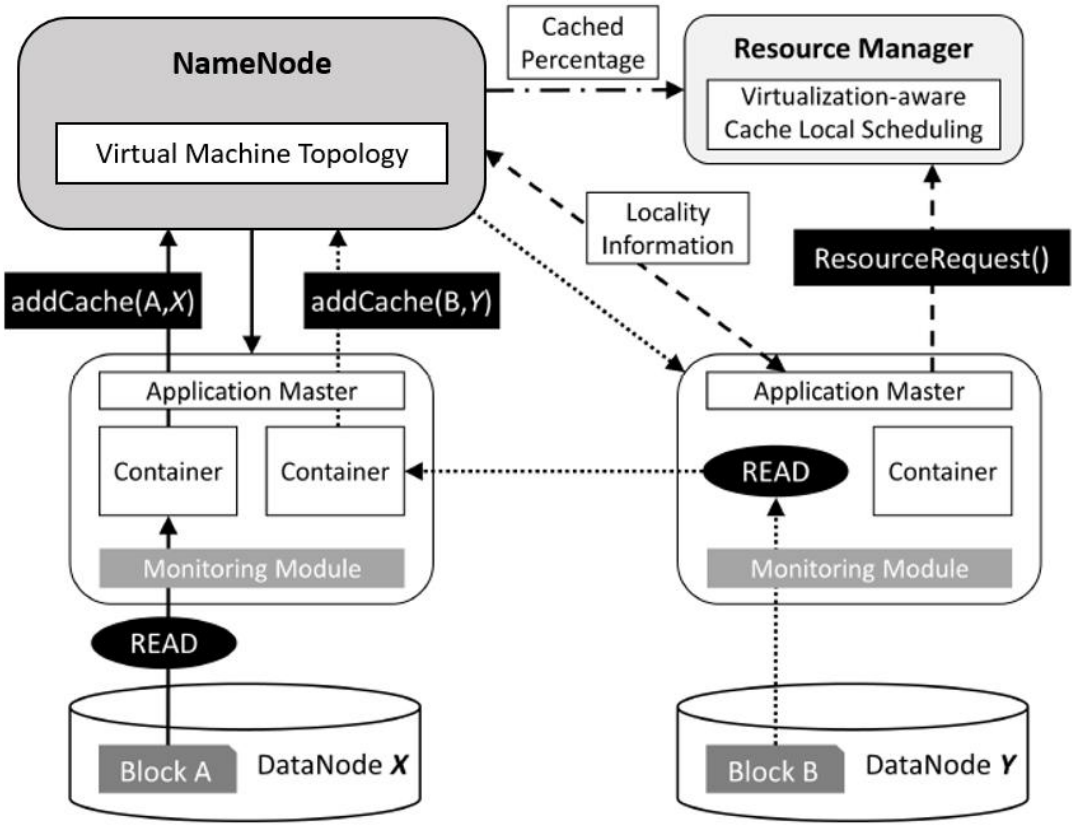

Figure 1. Architecture overview

CAVA let the NameNode store information about locations of every cached block and manage the topology of VMs. To cache an input block, in the original Hadoop, a user should select the path of input file manually by using a command [5, 9]. This command makes a request and sends it to the NameNode. The NameNode let the DataNode store an input block in OS page-cache. The DataNode sends the information about blocks in local disk periodically.

The problem of in-memory cache management system of the original Hadoop is that the user should manually select the file they want to store in memory [5]. In the same way, a user should select a file to evict specific files from in-memory cache.

To utilize the in-memory cache more effectively, we make Hadoop cache and evict the input block automatically. In CAVA, a Hadoop container sends cache request to the NameNode whenever they try to read the input block. We let the DataNode store the input block recently accessed. if the input block is cached in OS page cache, the DataNode calls mmap() and mlcok() system-calls to pin the input block. This method reduces caching overhead. In the original Hadoop, The NameNode does not know which

input block is probably cached in OS page cache and the Hadoop container does not send a caching request.

Figure 1 illustrates that two tasks in the DataNode X access one input block in local disk and access another input block in remote server Y. Each task which read these two input blocks sends the caching request to the NameNode. The task which reads an input block A sends the caching request to the NameNode (i.e. addCache(A,X)). Another task also sends the caching request for an input block B to the NameNode (i.e. addCache(B, Y)). After then, the NameNode determines whether it caches these input blocks in DataNodes or not. If the NameNode decides to cache both input blocks, the NameNode sends the caching command for the input block A to the DataNode X and for the data block B to the DataNode Y. Note that, although both tasks run on the DataNode X, the input block B is cached in the DataNode Y so that we can reuse OS page cache.

In CAVA, a MapReduce job is scheduled while the ApplicationMaster communicates with the NameNode and ResourceManager. The Application Master queries whether each input block is cached or not and the location of the input block. And the ApplicationMaster categorize multiple resource requests of the MapReduce application following fine-grained data localities which consider the VM topology and the in-memory cache. After then, it sends resource requests to the ResourceManager.

3. Virtualization-Aware Data Locality

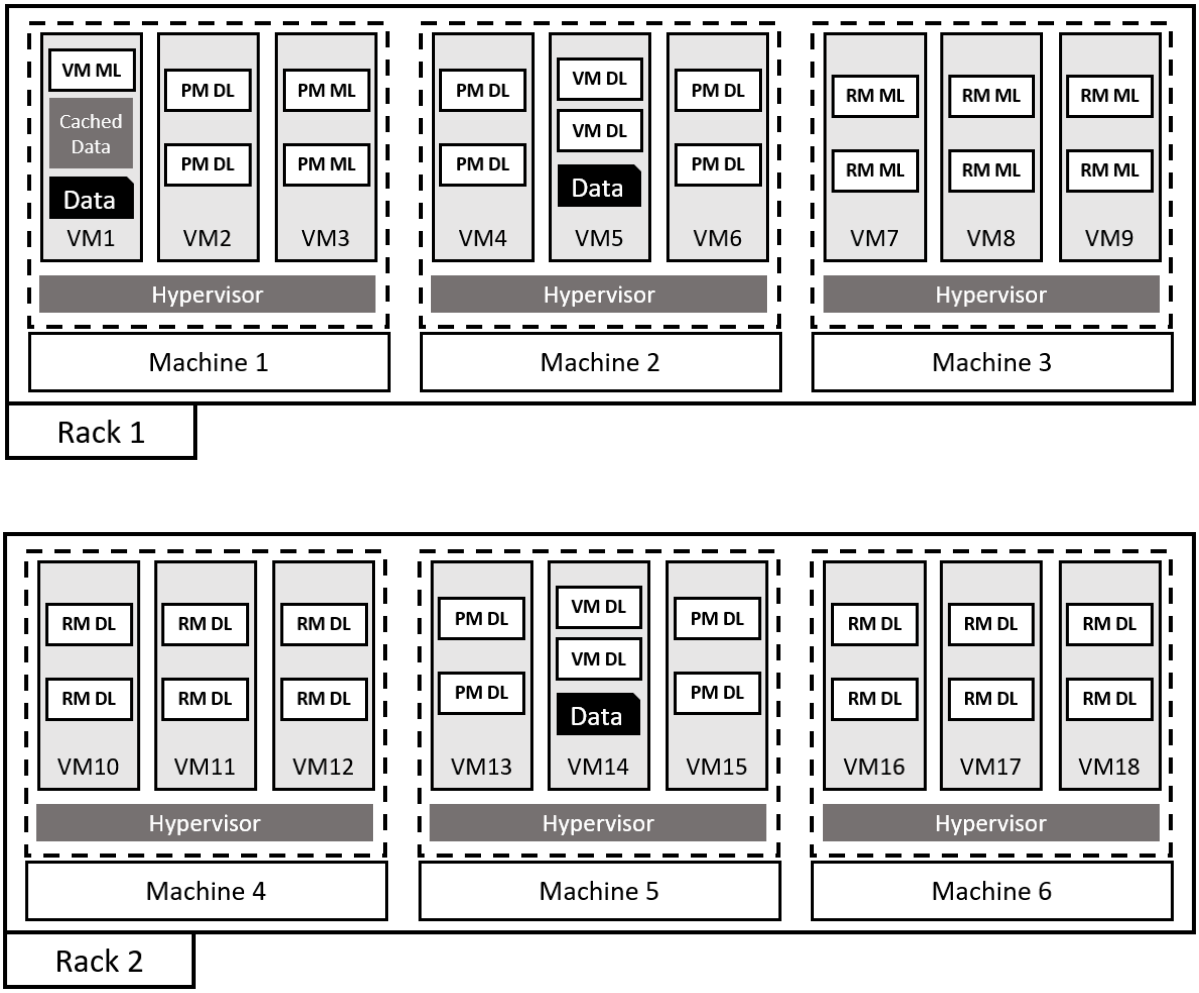

Figure 2. Fine-grained data locality in virtualized environment

Figure2 explains extra data localities in a virtualized environment. The original Hadoop provides only three kinds of data localities and it tries to schedule a task to a VM which has an input block of the task in its disk. If it is not possible to assign the task with the *data-local*, it tries to run the task on the other VM which does not have the input block but exist in same rack (*rack-local*). If every rack local nodes are busy, it assigns the task to a VM which is out of rack (*off-rack*).

Legacy localities do not consider the virtualized environment and the in-memory cache. The VM topology must be considered to set priorities of data localities accurately. In the virtualized cluster, transferring data between two VMs which is in the same physical machine does not use network bandwidth [6]. The CAVA improves the performance of the MapReduce application in the virtualized cluster by using this property.

The data locality we can find between different physical machine is also important. If the cluster is composed of many physical machines. It will have smaller chance to assign a task in the co-resident VM. In this study, we also investigate the data locality we can find between different physical machines but in the same rack.

Suppose that, VM1 and VM5 stores replicated input block in local disk of each VM and VM1 stores an input data in the in-memory cache. The Resource Manager should assign the task to VM1 so that the task runs ideally by reading cached input data from local VM. We call this locality *virtual machine memory locality*(VM-ML).

If the task has rare chance to be scheduled with VM-ML, the ApplicationMaster can consider VM5 which has the input block only in local disk. We call this locality as *virtual machine disk locality*(VM-DL). VM-DL is the same concept in legacy *data-local* but we used a different name to distinguish it from other fine-grained data locality levels.

Though VM5 has the input block in local disk, selecting VM2 is a better choice compared to choosing VM5. In the virtualized environment, the task can avoid network communications when it reads the input block from co-resident VM and can avoid expensive disk IO when the task read the input data from the in-memory. Though VM2 does not have the input block in its disk or the in-memory, VM2 still can read cached input block from VM1. We refer to this *physical machine memory locality*(PM-ML). Figure3 shows that reading data from the in-memory of co-resident VM is faster than reading data from local disk because reading data from co-resident VM does not use both network I/O and disk I/O.

If no VM with PM-ML is available, VM7 can be next option. Though VM7 and VM1 exist in different physical machine VM7 still can read the input data from in-memory of VM1. We refer to this *rack machine memory locality*(RM-ML). Figure3 shows that RM-ML is as fast as PM-ML. Because RM-ML can avoid disk I/O though it uses network bandwidth. In common case, matching VM-DL is more difficult than RM-ML, because the number of the RM-ML VMs is much bigger than VM-DL VMs when there is any cached input block in the same rack. So, RM-ML can also decrease scheduling delay for the task. Because of the above reasons, RM-ML is more effective than VM-DL.

If it is hard to assign the task with above localities. The Resource Manager can consider VMs like VM13 or VM15 which does not have input block either in a local disk or local in-memory but a disk of co-resident VM (VM14). If the task runs in VM13, it will read data from the disk of VM14 and use disk I/O but will not use network bandwidth. We refer to this locality as *physical machine disk locality*(PM-DL).

A VM like VM10 or VM11 cannot read the input block from the in-memory cache in the same rack but can read the input block from the disk of VM14 which is in the same rack but in the different physical machine. We refer to this as *rack machine disk locality*(RM-DL).

4. Overhead of Accessing Co-resident VM's In-Memory Cache

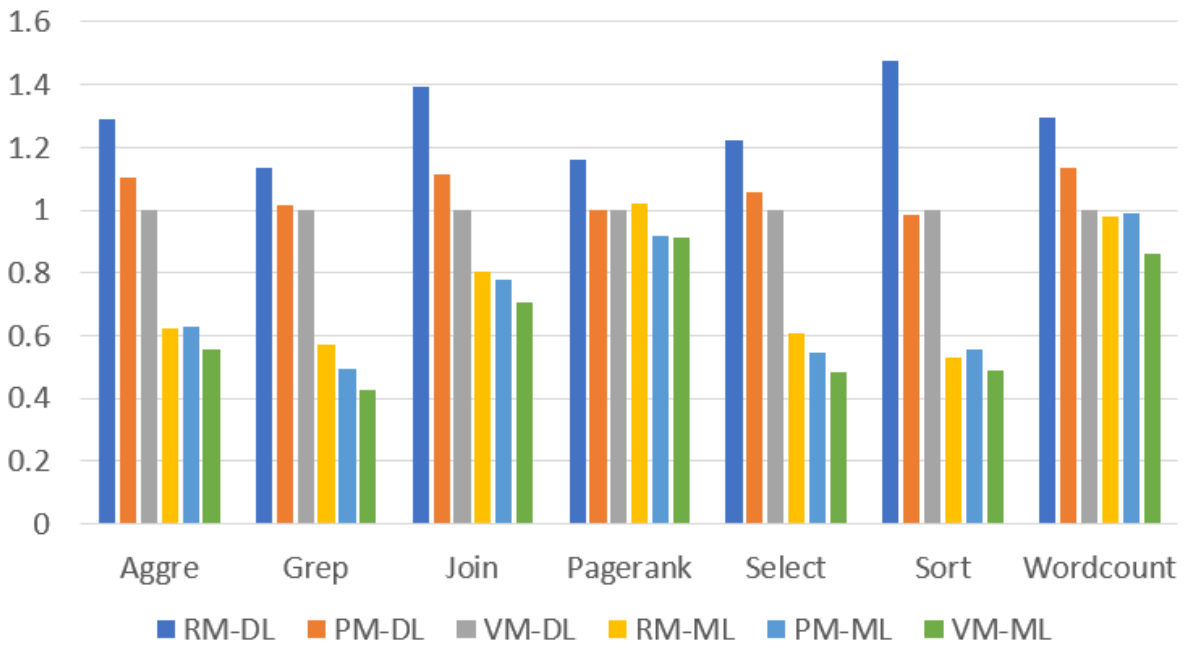

Figure 3. Performance comparison through various data locality

To evaluate how much fine-grained locality improves performance we ran one representative MapReduce application Grep. We used the machine which has 32 GB memory, Intel Xeon CPU E5-2640 32 core CPU. Size of input data was 8GB. In test machine, we assign 8 VMs with 3 virtual CPU and each VM ran the NodeManager and DataNode. We used distributed in-memory cache feature in the original Hadoop [5] to evaluate performance improvement of fine-grained locality levels which consider in-memory cache. To make every VM have every input block in in-memory of each VM, we set replication factor of HDFS as 8. And we set replication factor also 8 so that every DataNode stores every input in the local in-memory cache. In this way, we made the ideal case which every task reads their input block from the local in-memory cache without disk access. This experiment can disregard scheduling delay because every VM has input data.

Figure 3 shows the normalized total execution times of map tasks over multiple MapReduce applications with VM memory locality against VM disk locality. In our experiment, we make reduce task only run after map task finished.

To measure better utilization of in-memory cache. We evaluate performance improvement of PM-ML. We separated the NodeManager and DataNode. One VM runs the DataNode with 1 virtual CPU, another VM runs the NodeManager with 2 virtual CPU. and we set cache replication factor as 4. So, each DataNode stores every input blocks of the input file in the local in-memory cache. but Map task is run in the NodeManager VM. The NodeManager must read input data from the DataNode VM.

To evaluate the performance of RM-ML, we set two instances of Hadoop clusters across the two different physical machines. Each cluster runs the DataNode and NodeManager in the different physical machine. One physical machine runs set of DataNodes for cluster A and set of NodeMangers for cluster B and another physical machine runs set of DataNodes for cluster B and set of NodeMangers for cluster A. And each cluster runs the same MapReduce application with the same Input files. In this way, each cluster can run MapReduce application with rack machine locality while it generates the same amount of contention as the one generated in the experiment for evaluating overhead of physical machine locality. Each cluster uses the same configuration we used for PM-ML. The only difference is that the NodeManager and DataNode are on the different physical machine.

Figure3 shows that PM memory local scheduling is slightly slower than VM memory local scheduling and RM memory local scheduling is slightly slower than PM memory local scheduling in terms of the runtime of map task. Performance of VM memory locality, PM memory locality, and RM memory locality are improved 36.7%, 29.8% and 26.6% than VM disk locality in map task on average. This result implies that reading data from the in-memory of different VM in the same rack is slightly slower than reading data from local in-memory. Note that if an input block is cached in a VM in the same rack a task can take advantage of the in-memory cache.

PM disk locality is better than RM disk locality. Because it does not use network bandwidth. If an input block is in a disk of co-resident VM, a task earns benefits from PM disk locality through the replica of the input block which cannot read from local disk.

5. Fine-grained Memory Local Scheduling Algorithm

Algorithm 1 Fine-grained Memory Local Scheduling Algorithm

```
input: M_VM, // max skip for VM memory locality
       M_PM, // max skip for PM memory locality
       D_VM, // max skip for VM disk locality
       D_PM // max skip for PM disk locality
       M_RM // max skip for RM memory locality
initialize j.skip to 0 for each job j
when a heartbeat is received from VM n:
if n has a free slot then
   sort jobs in increasing order of number of running tasks
   for j in jobs do
      get current percentage of cached input data R_j for j
      if j has a VM memory local task t on n then
         launch t on n
         set j.skip = 0
      else if j has a PM memory local task t on n and j.skip ≥ R_j × M_VM then
         launch t on n
         set j.skip = 0
      else if j has a RM memory local task t on n and j.skip ≥ R_j × M_PM then
         launch t on n
         set j.skip = 0
      else if j has a VM disk local task t on n and j.skip ≥ R_j × M_RM then
         launch t on n
         set j.skip = 0
      else if j has a PM disk local task t on n and j.skip ≥ D_VM then
         launch t on n
         set j.skip = 0
      else if j has unlaunched task t and j.skip ≥ D_PM then
         launch t on n
      else
         set j.skip = j.skip + 1
      end if
   end for
end if
```

Figure 4. Fine-grained Memory Local Scheduling Algorithm

Figure 4 represents our scheduling algorithm which aware physical machine memory and rack machine memory locality. More fine-grained data locality level does not always guarantee effective utilizing of computing resources. So, our scheduling algorithm expands the existing delay scheduling. Our algorithm tries to schedule resource in VM memory local node first for job j. If VM memory local node is busy, it tries PM memory local node next. If VM-ML and PM-ML are not available then it tries RM memory locality. After every attempt fails for above memory localities scheduler tries VM-DL, PM-DL, and RM-DL in order.

Like the well-known delay scheduling [7]. Our scheduler tries higher level data localities first. After then, it tries lower level data localities. For several locality levels which use in-memory cache i.e. VM-ML, PM-ML, and RM-ML, we dynamically adjust the number of attempts proportional to the amount of cached input data. If the amount of the cached input data is small, the number of attempts to meet VM ML, PM-ML and RM-ML become small adaptively. In CAVA, the NameNode stores information about the amount of cached input block for each application and sends this information to the ResourceManager.

6. VM Topology

A Virtualized cluster has their own VM-topology including private cloud clusters. But, in most case, this information is not accessible by a user. On Amazon EC2, the user cannot figure out which VMs are running on which physical machine unless the user uses dedicated host or can put VM in specific host manually [1]. In this case, CAVA needs to find the VM topology by using traceroute from each VM to other VMs in the cluster and checking the first hop [6, 18].

7. Cache Replacement Algorithm

To manage scarce in-memory more efficiently, we implemented cache replacement policy in the NameNode. A cached input block whose cache affinity is low can be evicted when free in-memory space is not enough to cache another newly requested input block which has higher cache affinity.

We calculated the cache affinity of each application using offline profiling [13]. We measure a runtime of each MapReduce application by using both legacy Hadoop which does not use in-memory cache feature and CAVA which uses in-memory cache with the cache replication factor=1. The cache affinity is determined as $\mathrm{MAX}(1-\frac{R_{ML}}{R_{No-Cache}})$ where R_{ML} is the runtime of application with the legacy Hadoop, $R_{No-Cache}$ is the runtime with CAVA.

If TTL (Time to Live) time has been passed since the last access time, the cached input block is evicted with the highest priority when an incoming caching request requires more in-memory cache space but cluster does not have enough in-memory cache space.

Application	Cache Affinity
Grep	0.377
Aggregation	0.133
Join	0.073
Select	0.237
Wordcount	0.198
Sort	0.146

Table 1. Measured cache affinity of each MapReduce Applications

8. Evaluation

8.1. Methodology

In our experiments, we used 4 physical machines connected via 1 GE switch. Each physical server has four Intel Xeon Octa-core E5-4610v2 processors and 128 GB memory. For each server. One 7.2 K RPM HDD is used for VMs. We used 4.5.2 version of Xen hypervisor and 3.19.0-25-generic version of Linux kernel is installed in hosted Virtual Machine.

Our virtualized cluster is composed of 104 VMs including master VM. Master VM runs the ResourceManager and NameNode. And 103 slave VMs run the NodeManager and DataNode. Each physical machine contains 26 VMs. The master VM has 2 virtual CPUs and 6 GB memory. And slave VM has 1 virtual CPU and 3.8 GM memory. In each slave VM, The NodeManager occupies 2 GB memory and 1.5 GB memory is assigned for in-memory cache. Thus, the total amount of the in-memory cache in the virtualized cluster is 154.4 GB. Each map and reduce tasks use 2GB memory when they run. For HDFS, the block size is 64 MB, and the number of the replica is three. We implemented CAVA on the Hadoop 2.4.1 [4].

In our experiments, each workload consists of four applications, and thus, a quarter of the MapReduce cluster resources is used to execute each MapReduce applications. Table 2 shows MapReduce applications we used in our experiments and their input sizes. Aggregation, Join, and Selection are generated by HiveQL query (as in [10]) and every application uses different input files with each other and the total amount of the input files of each workload is smaller than the capacity of total in-memory cache in the MapReduce cluster.

Table 1 shows the list of the cache affinity for each MapReduce applications. To measure the cache affinity, we run MapReduce using two different version of Hadoop. One is the Hadoop which does not use the in-memory cache feature. Another is the CAVA which use the in-memory cache feature. After we measure a runtime of MapReduce application by using those two different versions of Hadoop for each application, we calculated cache affinity as we noted in Section 7.

For each experiment run, we start to execute applications in each workload without cached input data and we uninterruptedly run each application at least three times. The number of runs for each application depends on the runtime of each application.

We evaluate the performance of CAVA resource manager which considers fine-grained data locality and replaces cached input block based on the cache affinity of MapReduce applications. And we compare its performance with the following versions.

- Hadoop No-Cache: The Baseline Hadoop which does not use the in-memory caching feature when MapReduce application runs and only considers VM disk locality.

- No P/RM-ML: This version of Hadoop behaves exactly same way with CAVA but Resource Manager does not consider PM-ML and RM-ML when it schedules resource for a task.
- No RM-ML: This version of Hadoop behaves exactly same way with CAVA but Resource Manager does not consider RM-ML when it schedules resource for a task.

For No-Cache, the minimum number of skips for VM disk locality, D_{VM} is set to 103. For CAVA and No RM-ML, the minimum numbers of skips for VM memory, PM memory, RM Memory, VM disk, and PM disk, RM disk localities, M_{VM}, M_{PM}, M_{RM}, D_{VM}, D_{PM}, D_{PM} and D_{RM} are set to 0, 3, 13, 26, 52, 78 respectively. For No P/RM-ML which is not able to consider PM-ML and RM-ML, D_{VM} is set to 13.

8.2. Experimental Result

Workload	App1	App2	App3	App4	Total Input (GB)
W1	Grep	Aggregation	Join	Wordcount	284.8
W2	Grep	Aggregation	Join	Selection	264.9
W3	Grep	Join	Pagerank	Selection	206.6

Table 2. Workloads composed of multiple MapReduce applications

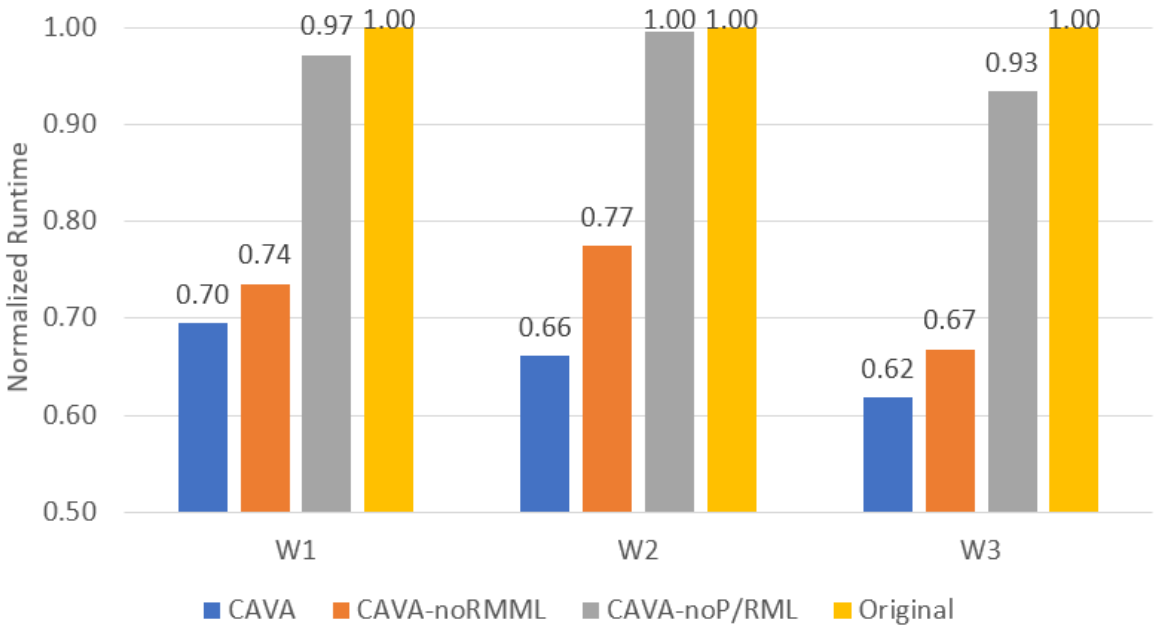

Figure 5. Normalized total runtimes of two workloads from the second run

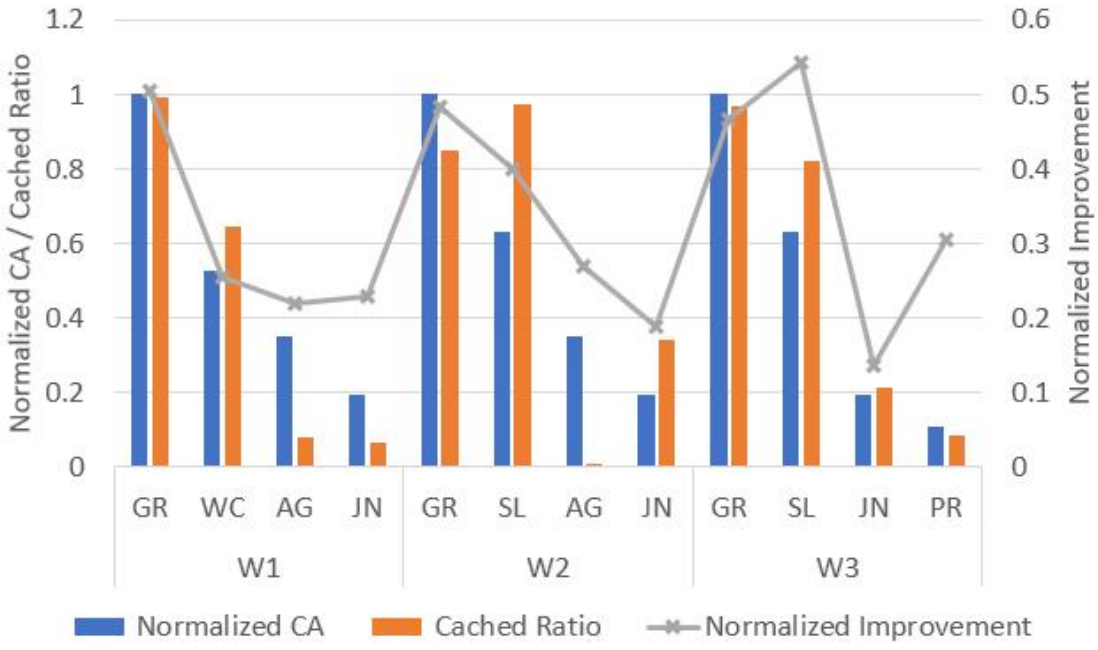

Figure 6. Normalized runtime, Cached Ratio and Normalized Improvement of each application from the second run

Figure 5 shows the total runtime of each workload, normalized to that with No-Cache. For the result, we do not include execution time of the first run where the in-memory cache is not warmed up yet. Compared to the No-Cache, the performance improvement of the No-R/PM-ML, No-RM-ML, and CAVA over four workloads are 96.7%, 72.6% and 65.8% on average, respectively. From the results, we can see that applying PM-ML and RM-ML provides 34.2% performance improvement on average. The contribution of the PM Memory locality and the RM Memory locality on performance is 24.1% and 6.8% on average, respectively. Because the PM memory locality has higher priority than the RM memory locality so a task gets more chance to be allocated with the PM memory locality than the RM memory Locality. So, the PM memory locality provides more benefits than the RM memory locality.

Figure 5 presents shorten runtimes of MapReduce applications with the CAVA which are normalized to those with No-Cache for each workload. Figure 5 also presents a ratio of the cached input data and cache affinity which is normalized to the cache affinity of Grep which has the highest cache affinity among all applications in our experiment.

MapReduce applications whose cache affinity is relatively higher than others, like Grep, Selection have a large fraction of cache. In these results, the runtimes of MapReduce application whose cache affinity is higher than others like Grep, Selection are significantly reduced. The runtimes of MapReduce applications whose cache affinity is relatively low like Join and Aggregation are also decreased because the bottleneck from disk I/O is mitigated by reusing the cached input block even if those are in other VM.

9. Conclusion

In this work, we investigated extra levels of locality of an input data in a virtualized cluster for big data analytic frameworks. We designed and implemented the Fine-grained locality-aware resource scheduling system on top of Hadoop. Our system considers the physical machine memory locality and rack machine memory locality so that MapReduce task leverage extra levels of data locality in the virtualized cluster. Even if the data transferring between two different physical machine uses network bandwidth, reusing the cached input data in other physical machine is faster than reading the input data from the local disk. Our experimental results on a cluster which consist of 104 VM and uses 4 physical machines shows that CAVA system improves the performance of various workloads which consist of multiple MapReduce application by 34.2% on average compared to legacy Hadoop which does not use the in-memory caching feature and not consider the virtualized environment.

10. Related Work

Several techniques to improve the performance of MapReduce applications in clouds have been studied [6,14,12].

Techniques to achieve the data locality of an input block on executing tasks are investigated [15, 16].

There is research about using an input data from local storage like our PM disk locality. But locality of the in-memory locality is not considered [6].

The legacy data locality that the original Hadoop provides is not enough for Hadoop which uses the in-memory caching feature. To solve this problem, the in-memory cache locality that considers the location of a cached input block has been investigated in recent literature [13]. However, additional levels of data locality which considers a virtualized environment, i.e. PM-ML, and RM-ML are not supported when it schedules tasks.

Several techniques to handle a data-intensive job such as Quincy [11] have been proposed to consider data locality to improve the performance.

There are some efforts to achieve better utilization of the in-memory cache and data locality to improve performance [2]. PACMan is a coordinated management system for distributed in-memory caches. PACMan provides memory locality for multiple parallel tasks to improve the performance [3]. Its cache eviction policy evicts data blocks from large incomplete inputs based on “all-or-nothing” property. This approach can be effective for short jobs which can run all their tasks in parallel. PACMan allows the unlimited number of cache replicas for a block to increase the probability to achieve memory locality.

Spark [24] is a framework to implement Resilient Distributed Datasets (RDDs) [23]. As in-memory data object, RDD stores intermediate data and reused by multiple Bigdata analytic applications.

REFERENCES

[1] Amazon ec2 dedicated hosts. https://aws.amazon.com/ec2/dedicated-hosts/.

[2] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Disk-locality in datacenter computing considered irrelevant. In Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems, HotOS'11, pages 12–12, Berkeley, CA, 2011. USENIX Association.

[3] G. Ananthanarayanan, A. Ghodsi, A. Wang, D. Borthakur, S. Kandula, S. Shenker, and I. Stoica. Pacman: Coordinated memory caching for parallel jobs. In Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, NSDI'12, pages 20–20, Berkeley, CA, 2012. USENIX Association.

[4] Apache hadoop. http://hadoop.apache.org/.

[5] Apache hadoop centralized cache management in hdfs. http://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-hdfs/CentralizedCacheManagement.html.

[6] X. Bu, J. Rao, and C.-z. Xu. Interference and locality-aware task scheduling for mapreduce applications in virtual clusters. In Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing, HPDC '13, pages 227–238, New York, NY, USA, 2013. ACM.

[7] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In Proceedings of the 5th European Conference on Computer Systems, EuroSys '10, pages 265–278, New York, NY, 2010. ACM.

[8] C. Delimitrou and C. Kozyrakis. Quasar: Resourceefficient and QoS-aware cluster management. In 11 Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2014.

[9] A. Wang and C. McCabe. In-memory caching in hdfs: Lower latency, same great taste. In Hadoop Summit, 2014.

[10] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. De- Witt, S.Madden, andM. Stonebraker. A comparison of approaches to large-scale data analysis. In Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, SIGMOD '09, pages 165−178, New York, NY, 2009. ACM.

[11] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: Fair scheduling for distributed computing clusters. In Proceed- ings of the ACMSIGOPS 22Nd Symposium on Operating Systems Principles, SOSP '09, pages 261–276, New York, NY, 2009. ACM.

[12] J. Park, D. Lee, B. Kim, J. Huh, and S. Maeng. Locality-aware dynamic vm reconfiguration on mapreduce clouds. In Proceedings of the 21st Inter- national Symposium on High-Performance Parallel and Distributed Computing, HPDC '12, pages 27– 36, New York, NY, USA, 2012. ACM.

[13] J. Kwak, E. Hwang, T. K. Yoo, B. Nam, and Y. R. Choi. In-memory caching orchestration for hadoop. In 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pages 94–97, May 2016.

[14] B. Palanisamy, A. Singh, L. Liu, and B. Jain. Purlieus: Locality-aware resource allocation for mapreduce in a cloud. In Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, pages 58:1–58:11, New York, NY, USA, 2011. ACM.